Digital Signal Processing Filtering with GPU

Sajid Anwar, Wonyong Sung

School of Electrical Engineering
Seoul National University
Gwanak-gu, Seoul 151-744 Korea
sajid@dsp.snu.ac.kr
wysung@snu.ac.kr

Abstract— **This work designed digital filter kernels with GTX 260 GPU. Various FIR and IIR kernels are implemented. IIR filters due to its recursive nature have dependency on previously computed output samples. The dependency problem can be solved by separate computation of particular and homogeneous solutions. Comparing to CPU based implementation experimental results show speed improvement of 3 to 40 times for IIR and FIR filter kernels respectively.**

***Keywords*— FIR, IIR, GPU, CUDA, Homogenous Solution, Particular Solution, GPGPU**

I. Introduction

Graphics Processing Unit (GPU) is responsible for manipulating and displaying graphical data. Present GPUs have hundreds of CPU cores and are thus excellent in parallel processing. Each of the CPU core can run hundreds of threads in parallel. This huge amount of processing capability leads to General Purpose Computing on GPU (GPGPU). However GPGPU limits the developer to a set of defined graphics API. The developer has to conduct the task using standardized graphics APIs like OpenGL and DirectX. With the advent of Compute Unified Device Architecture (CUDA), these limitations have vanished [2]. This work has implemented both recursive and non-recursive filters on GPU with CUDA. In contrast to FIR filters, recursive filters have dependency on previous output samples. A technique is used to reduce the dependency time. Nvidia GTX 260 having 194 cores is the implementation platform.

The next two sections discuss FIR and IIR kernels and corresponding CUDA implementations. Section 4 discusses mapping of our suggested solution onto CUDA threads and blocks. Section 5 finally concludes.

II. FIR Filter Kernels

FIR filters have inherent parallel nature and are easily implemented with CUDA. A 16-tap FIR filter is implemented. Fast shared memory on the GPU is intelligently utilized to save valuable cycles. Profiling results are given in Table 3. The consumed time includes memory transfers as well.

III. IIR Filter Kernels

A first Order recursive filter is shown in Fig. 1. It's evident from the figure that the current output is dependent on current input sample and previous output sample. Dependence on previous output sample describes the inherent sequential nature of this category of filters. The dependency time can be reduced by decomposing the computation into two separate solutions; Homogenous and Particular [1]. Figure 2 shows this decomposition process. It can be observed that the homogeneous solution is dependent on initial condition while particular solution is not. So particular solution can now be computed in parallel. The next section discusses how this scheme is mapped onto the GPU.

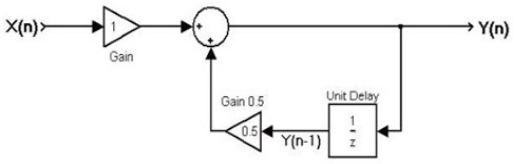

Figure 1 First Oder IIR Filter

$$y[n] = ay[n-1] + x[n]$$
$$y[n+1] = ay[n] + ax[n+1]$$
$$y[n+1] = a^2 y[n-1] + ax[n] + x[n+1] \text{ similarly}$$
$$y[n+2] = a^3 y[n-1] + a^2 x[n] + ax[n+1] + x[n+2]$$
$$y[n+p] = a^{p+1} y[n-1] + a^p x[n] + \ldots + a^{p-2} x[n+2] + \ldots$$
$$a^{p+1} y[n-1] \text{ Homogeneous soultion}$$
$$z[n] = a^p x[n] + a^{p-1} x[n+1] + a^{p-2} x[n+2] +$$
$$\ldots \text{ Particular Solution}$$

Figure 2 Decomposition Process

IV. Mapping Onto GPU

Particular solution is computed in parallel by a total of L blocks. Each block contains M threads. Each thread process N input elements sequentially. All threads run in parallel. Table 2 and Fig. 3 shows this process. Threads are synchronised at their finishing point. Once this is done thread level initial condition is propagated as depicted by Fig. 4. This propagation is sequentially conducted. At the end of step 2 (Fig. 4), each block contains particular solution.

Table 2: Mapping of input data structure to CUDA blocks and threads

	0th thread	1st thread	...	(M-1)th thread
0th block	x[0,0,0] to x[0,0,N-1]	x[0,1,0] to x[0,1,N-1]	...	x[0,M-1,0] to x[0,M-1,N-1]
1st block	x[1,0,0] to x[1,0,N-1]	x[1,1,0] to x[1,1,N-1]	...	x[1,M-1,0] to x[1,M-1,N-1]
...	...	...	...	...
(L-1)th block	x[L-1,0,0] to x[L-1,0,N-1]	x[L-1,1,0] to x[L-1,1,N-1]	...	x[L-1,M-1,0] to x[L-1,M-1,N-1]

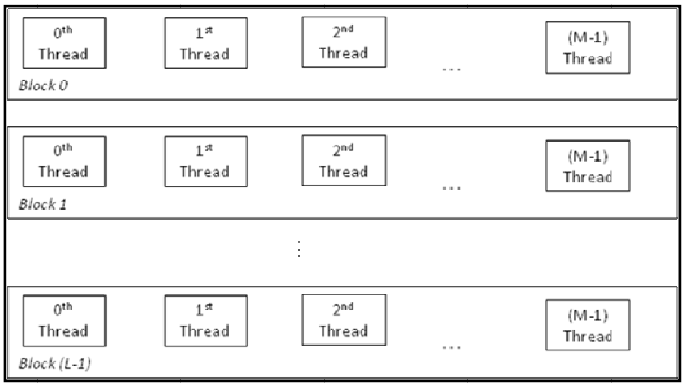

Figure 3 Particular Solution by each block (Step 1)

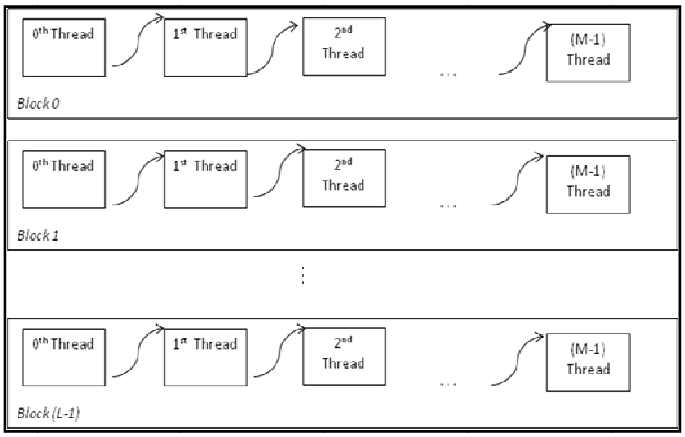

Figure 4 Propagating Particular Solution (Step 2)

The final step is to propagate block level initial condition and sum up the results. This means block level synchronization is needed at this point. Blocks are intercommunicated by transferring data to device global memory. Figure 5 shows timing analysis per output sample. Time for one output can be reduced by increasing *L*, which is the number of threads per block. Further higher number of threads per block may reduce number of blocks and hence the expenses of block level synchronization. P represents processor cores. Table 3 shows the profiling results. It is believed that these results can further be improved with intelligent memory hierarchy utilization.

Table 3 Performance Results

Category	Input (million)	GPU time (µs)	CPU time (µs)	Ratio
16 Tap FIR	1	2654	103861	39.133
IIR	2.5	5134	13030	2.5
IIR	4.9	9431	28200	3

$$
\begin{aligned}
&TimeforOneOutput = T_{PL} / PL \\
&= (2LMul + 2LAdd + PMul + PAdd + Psync) / PL \\
&= (2Mul + 2Add) / P + (Mul + Add) / L \\
&= (2Mul + 2Add) / P + (Mul + Add) / L
\end{aligned}
$$

Figure 5 Time per Output Sample

V. Conclusion

This work explored implementation of digital filters with GPU. Recursive filters are efficiently implemented by decomposing solution into particular and homogeneous solution. It is observed that block level synchronization is expensive and can be reduced by increasing thread level parallelism. Further the analysis showed that increasing number of threads per blocks reduces time pre output sample. GPU based implementation proved to be 3 to 40 times efficient than alternative CPU implementation. Future work is to efficiently utilize the memory hierarchy for recursive filters.

Acknowledgement

This work is supported in part by the Ministry of Education and Human Resources Development (MOEHRD) under the Brain Korea (BK21) project and in part supported by Higher Education Commission (HEC) Pakistan.

References

[1] W. Sung and S. K. Mitra, "Efficient Multi-Processor Implementation of Recursive Digital Filters," ICASSP 1986

[2] Dr. Dobb, "Supercomputing for the masses," [Available Online] http://www.ddj.com/hpc-high-performance-computing/207402986